

SSSSSSSSSSSSSS	000000000	RRRRRRRRRRRR	TTTTTTTTTTTTTT	3333333333	2222222222
SSSSSSSSSSSSSS	000000000	RRRRRRRRRRRR	TTTTTTTTTTTTTT	3333333333	2222222222
SSSSSSSSSSSSSS	000000000	RRRRRRRRRRRR	TTTTTTTTTTTTTT	3333333333	2222222222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSSSSSSSSS	000	RRRRRRRRRRRR	TTT	333	222
SSSSSSSSSS	000	RRRRRRRRRRRR	TTT	333	222
SSSSSSSSSS	000	RRRRRRRRRRRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSS	000	RRR	TTT	333	222
SSSSSSSSSSSS	000000000	RRR	TTT	3333333333	22222222222222
SSSSSSSSSSSS	000000000	RRR	TTT	3333333333	22222222222222
SSSSSSSSSSSS	000000000	RRR	TTT	3333333333	22222222222222

```
SSSSSSSS 000000 RRRRRRRR LL      IIIIII 88888888
SSSSSSSS 000000 RRRRRRRR LL      IIIIII 88888888
SS        00      00 RR        RR LL      II      88      88
SS        00      00 RR        RR LL      II      88      88
SS        00      00 RR        RR LL      II      88      88
SS        00      00 RR        RR LL      II      88      88
SSSSSS    00      00 RRRRRRRR LL      II      88888888
SSSSSS    00      00 RRRRRRRR LL      II      88888888
          SS 00      00 RR  RR   LL      II      88      88
          SS 00      00 RR  RR   LL      II      88      88
          SS 00      00 RR  RR   LL      II      88      88
          SS 00      00 RR  RR   LL      II      88      88
SSSSSSSS 000000 RR        RR LLLLLLLLLL IIIIII 88888888
SSSSSSSS 000000 RR        RR LLLLLLLLLL IIIIII 88888888
                                     ....
                                     ....
                                     ....
                                     ....
```

```
RRRRRRRR EEEEEEEEE QQQQQQ
RRRRRRRR EEEEEEEEE QQQQQQ
RR        EE        QQ      QQ
RR        EE        QQ      QQ
RR        EE        QQ      QQ
RR        EE        QQ      QQ
RRRRRRRR EEEEEEEEE QQ      QQ
RRRRRRRR EEEEEEEEE QQ      QQ
RR  RR   EE        QQ      QQ
RR  RR   EE        QQ      QQ
RR  RR   EE        QQ      QQ
RR  RR   EE        QQ      QQ
RR  RR   EE        QQ      QQ
RR  RR   EEEEEEEEE QQQQ  QQ
RR  RR   EEEEEEEEE QQQQ  QQ
```

File: SORLIB.REQ IDENT = 'V04-000' ! File: SORLIB.REQ Edit: PDG3034

COPYRIGHT (c) 1978, 1980, 1982, 1984 BY  
DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.  
ALL RIGHTS RESERVED.

THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED  
ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE  
INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER  
COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY  
OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY  
TRANSFERRED.

THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE  
AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT  
CORPORATION.

DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS  
SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.

++

FACILITY: VAX-11 SORT / MERGE

ABSTRACT:

This is the common definition file for VAX-11 SORT / MERGE.  
All definitions of interest to more than one module are in this file.  
This file is used as a library source.

ENVIRONMENT: VAX/VMS user mode

AUTHOR: P. Gilbert, CREATION DATE: 07-Dec-1981

MODIFIED BY:

T03-015 Original  
T03-016 Add section on pad characters, and correct the extension for  
specification files (.SRT). PDG 13-Dec-1982  
T03-017 Add WF\_NAMES, CFT indices of work file names. PDG 26-Dec-1982  
T03-018 Added DDB\_CHAN. PDG 28-Dec-1982  
T03-019 Make work-file description blocks (WFBs) distinct from DDBs.  
PDG 31-Dec-1982  
T03-020 Add clean-up routines. PDG 4-Jan-1983  
T03-021 Add WFB\_DEV. PDG 6-Jan-1983  
T03-022 Removed PT/ST\_ADR; added BS\_DECM, WRK\_SIZ. PDG 26-Jan-1983  
T03-023 Change STAT\_K\_WRK\_USE to STAT\_K\_WRK\_ALQ. Added WFB\_USE field.  
Added COM\_MRG\_STREAM for stable merges. PDG 27-Jan-1983  
T03-024 Remove section on pad characters. Add COM\_PAD. PDG 8-Feb-1983  
T03-025 Remove unreferenced fields. Change linkage declarations so



register information is available to SOR\$\$KEY.SUB at run time.  
Define the macro SOR\$\$FATAL. PDG 16-Mar-1983  
T03-026 Give the SOR\$RO\_CODE\_n PSECTs the EXE attr. PDG 7-Apr-1983  
T03-027 Information hiding of WFB structure. PDG 12-Apr-1983  
T03-028 Move definitions of fields specific to scratch-i/o to SORSCRIO  
from this module. PDG 18-Apr-1983  
T03-029 Reduce COM\_K\_SCRATCH. PDG 22-Apr-1983  
T03-030 Correct size of COM\_WF\_NAMES. PDG 17-May-1983  
T03-031 Add COM\_ARCHFLAG. PDG 31-Jan-1984  
T03-032 Add COLC\_BLOCK stuff. PDG 22-Feb-1984  
T03-033 Change TON\_K\_BUFSIZE to 5 blocks for VAXELN.  
Add support for VAXELN. Jeff East 3/13/84  
T03-034 Change COM\_RHB to COM\_RHB\_INP and COM\_RHB\_OUT.  
This is to avoid problems with merge, where an incoming  
record overwrites the VFC area for the outgoing record.  
PDG 24-Jul-1984

--  
LIBRARY 'SYSS\$LIBRARY:STARLET';  
LIBRARY 'SYSS\$LIBRARY:XPORT';

## X P O R T

The use of XPORT causes some problems, most notably with alignment, and the default sign extension. The following macros are used.

MACRO  
XBYTE = \$ALIGN(BYTE) %EXPAND \$BITS(8) %  
XWORD = \$ALIGN(WORD) %EXPAND \$BITS(16) %  
XLONG = \$ALIGN(FULLWORD) %EXPAND \$BITS(32) %  
XDESC = \$ALIGN(FULLWORD) %SUB BLOCK(2) %  
XADDR = \$ALIGN(FULLWORD) %ADDRESS %  
\$SHOW(FIELDS)

## POSITION AND SIZE MACROS

MACRO

: Macros used for field references

```

: A_ = 0, 0, 0 %
: L_ = 0, 32, 0 %
: BASE_ = 0, 0, 0 %

```

```

: Macros to construct a bit mask from a standard four-component field
: definition (offset, position, size, extension). The result has set
: bits in those positions that belong to the field. A list of field
: definitions can be specified.

```

: Example:

```

:     MACRO
:         A=0,2,4,0%,
:         B=0,9,1,0%

```

: MASK\_(A,B) is equal to %B'1000111100'

```

: XMASK [O,P,S,E]=
:   (1 ^ ((P)+(S))) - (1 ^ (P)) %,

```

```

: MASK []=
:   0 OR XMASK_ (%REMAINING)) %,

```

```

: Macros to align a specified value at the bit position specified by a
: standard four-component field definition (offset, position, size,
: extension). A list of values and field definitions can be specified.

```

: Example:

```

:     MACRO
:         A=0,2,4,0%,
:         B=0,9,1,0%

```

: ALIGN\_(7,A,1,B) is equal to 7^2 OR 1^9

```

: XALIGN [V,O,P,S,E]=
:   ((V) ^ (P)) %,

```

```

: ALIGN []=
:   0 OR XALIGN_ (%REMAINING)) %;

```

## G E N E R A L

## LITERAL

```
TRUE=      1;
FALSE=     0;
```

## MACRO

```
ELIF=      ELSE IF %;
```

## MACRO

```
! Macro to round a value to the next higher multiple of a number.
! The first parameter is the number which is to be rounded.
! The second parameter is the multiple up to which we round.
! If omitted, the default for the second parameter is %UPVAL.
! The second parameter should be a literal, and a power of 2.
```

```
ROUND (A,B) =
  %IF %NULL(B)
  %THEN (((A) + %UPVAL-1) AND NOT (%UPVAL-1))
  %ELSE (((A)+ (B) -1) AND NOT ((B) -1))
  %FI %;
```

## MACRO

```
! Macro to calculate floor(log2(constant))
```

```
LN2_(A)=
  (%NBITSU(A)-1) %;
```

## MACRO

```
! Macro to signal an internal consistency check.
```

```
BUGCHECK(A)=
  BEGIN BUILTIN CHMU;
  CHMU(%REF(0));
  0
  END %;
```

## MACRO

```
! Macro to establish a condition handler.
```

```
ESTABLISH_(X) =
  BEGIN BUILTIN FP;
  .FP = X;
  END %;
```

## MACRO

```
! Macro to produce a list of names
```

```
PREFIX_(A)[B] = %NAME(A,B) %;
```



## MACRO

! Macros to determine if the value of an expression is one of a set of specified small-integer values. These macros can be used only if the following conditions are met:

The value to be tested is in the range 0 through 127.

The values to be tested for are all in the range 0 through 31.

Example:

IF ONEOF(.X, BMSK\_(1,3,5)) ...

The code generated is much more efficient than a series of comparisons (provided that the parameters of BMSK\_ are all compile-time constant).

```
XBMSK [A]=
  %IF (A) GTRU 31 %THEN %WARN('ONEOF won't work') %FI
  (1 ^ (31 - (A))) %,
```

```
BMSK []=
  (0 OR XBMSK_(%REMAINING)) %,
```

```
ONEOF (A,B)=
  ((B) ^ (A)) LSS 0) %;
```

## MACRO

! Macros to create initialized, read-only bit-vectors.  
! The first parameter to BV\_ is the largest element which will be accessed in the bit-vector.

For example:

```
OWN PRIMES: BV_( 51, 2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,51 );
IF .PRIMES[.I]
THEN % ( I is Prime )%
ELSE % ( I is Composite )%
```

```
BV_1_[A] = [A] = 1 %,
```

```
BV_(M) = BITVECTOR[M+1]
PSECT(SOR$RO_CODE) PRESET( BV_1_(%REMAINING) ) %;
```

## MACRO

! Macros to distinguish whether the value of an expression is among one set of values, or another set of values, based on a single bit.  
! An error diagnostic is issued if a single bit will not suffice.

```
DIST (X,Y,Z) =
  BEGIN
  LITERAL
```



```
      M = (DIST1 (%REMOVE(Y)) XOR DIST1 (%REMOVE(Z))) AND NOT
            (DIST2 (%REMOVE(Y)) OR DIST2 (%REMOVE(Z))),
      L = %NBITSU(M XOR (M-1))-1;
%IF M EQL 0 %THEN %ERROR('Oops') %FI
%IF (DIST1 (%REMOVE(Y)) AND 1^L) EQL 0
%THEN      ((X) AND 1^L) EQL 0
%ELSE      ((X) AND 1^L) NEQ 0
%FI
END %;
DIST1_(X) = X %;
DIST2_(X)[ ] = (0 OR DIST3 (X,%REMAINING) + 0) %;
DIST3_(X)[Y] = (X XOR Y) %;
```

## DEBUGGING CODE

This section defines macros to aid in writing debugging code.

The %VARIANT switch is used to conditionally include compiler debugging code. When %VARIANT is true, debugging code is included. When it is false, debugging code is omitted. The macro DEB\_CODE is provided to bracket debugging code that is to be unconditionally executed.

In addition, the global variable "SOR\$\$D" in the COMENTRY module can be used to obtain conditional execution of debugging code. This variable is initialized to zero, but may be altered during the initial DEBUG dialogue, before the compiler is started:

```
DBG>D SOR$$D=%X'D6003FFF'      (for example)
DBG>D SOR$$D=1                  (for example)
DBG>G
```

The bits in the variable "SOR\$\$D" are allocated as follows:

0	%X'00000001'	Dump run information
1	%X'00000002'	Dump incremental statistics
2	%X'00000004'	Dump allocation information
30	%X'40000000'	Unassigned
31	%X'80000000'	Unassigned

The macro DEB\_SWITCH is provided to bracket conditionally executed debugging code.

## MACRO

! Macro to bracket unconditional debugging code. The parameter is an expression that will be compiled if %VARIANT is true.

```
DEB_CODE(A)=
%IF %VARIANT
%THEN
A
%FI %.
```

! Macro to bracket conditional debugging code. The first parameter is a bit number in the variable SOR\$\$D, and the second parameter is an expression that will be evaluated if that bit is set. The entire expansion is compiled only if %VARIANT is true.

```
DEB_SWITCH(A,B)=
%IF %VARIANT
%THEN
BEGIN EXTERNAL SOR$$D;
IF .SOR$$D<A,1> THEN B;
END
%FI %.
```

! Macro to test an assertion about compile-time constants.

```
! ASSERT (A)=  
!   XIF NOT (A)  
!   XTHEN  
!     XERROR('Assertion failed')  
!   XFI X;
```

## MAXIMUM VALUES

LITERAL

```
MAX_KEYS= 255,      ! Maximum number of sort keys allowed
MAX_FILES= 10,       ! Maximum number of input files.
MIN_WORK_FILES= 1,   ! Minimum number of work files
DEF_WORK_FILES= 2,   ! Default number of work files
MAX_WORK_FILES= 10,  ! Maximum number of work files
MAX_MERGE_ORDER= 10, ! Maximum merge order
MAX_SPC_LINE= 132,   ! Maximum length of spec file line

MAX_SEQ_RECLN= 32767, ! Maximum sequential file record length
MAX_REL_RECLN= 16384, ! Maximum relative file record length
MAX_IDX_RECLN= 16384, ! Maximum indexed file record length
MAX_ISAMKEYLEN= 255,  ! Maximum index key data item length
MAX_REFSIZE= 65535,   ! Maximum length of a referenceable data-item
MAX_PSECTSIZE= 2147483647; ! Maximum length of a PSECT
```

LITERAL

```
MIN_MBC= 7,          ! Minimum MBC count
MAX_MBC= 16,         ! Maximum MBC count (for RP06)
MIN_MBF= 0,          ! Minimum MBF count
MAX_MBF= 2,          ! Maximum MBF count
```

LITERAL

```
DEF_FILE_ALLOC= 128*3, ! Default file allocation
DEF_TRM_ALLOC= 16;     ! Default allocation for terminals
```

LITERAL

```
COM_K_BPERPAGE= 512,   ! Bytes per page
COM_K_BPERBLOCK= 512;  ! Bytes per disk block
```

LITERAL

```
! Define a literal for the amount of work space to allocate
! for specification text, and another for the amount of work space
! to allocate if we only need to process a collating sequence.

WRK_K_ALLOC= 128 * COM_K_BPERPAGE, ! Allocation for work area
WRK_K_COLLATE= 6 * 256;           ! Alloc to process collating sequence
```



## INTERFACE VALUES

## LITERAL

Datatype values for use in the key definition buffer (KEY\_BUFFER).

These are also used to define the global literals SOR\$GK\_XXX\_KEY.

These are used only for compatability purposes.

KEY_K_CHAR=	1.	Character data
KEY_K_BIN=	2.	Signed binary data
KEY_K_ZONE=	3.	Zoned decimal
KEY_K_PACK=	4.	Packed decimal
KEY_K_USB=	5.	Unsigned binary
KEY_K_DLO=	6.	Decimal leading overpunch
KEY_K_DLS=	7.	Decimal leading separate
KEY_K_DTO=	8.	Decimal trailing overpunch
KEY_K_DTS=	9.	Decimal trailing separate
KEY_K_FLT=	10.	Floating
KEY_K_FLTD=	11.	D_floating
KEY_K_FLTG=	12.	G_floating
KEY_K_FLTH=	13.	H_floating
KEY_K_MAX=	13.	Maximum

## LITERAL

Values for sort types, passed to SOR\$INIT\_SORT.

These are also used to define the global literals SOR\$GK\_XXX.

TYP_K_RECORD=	1.	Record sort
TYP_K_TAG=	2.	Tag sort
TYP_K_INDEX=	3.	Index sort
TYP_K_ADDRESS=	4.	Address sort
TYP_K_MAX=	4.	Maximum sort type

## MACRO

Options flags, passed to SOR\$INIT\_SORT and SOR\$INIT\_MERGE.

These are used to define the global literals SOR\$V\_XXX and SOR\$M\_XXX.

OPT_STABLE=	0, 0, 1, 0 X,	Stable sort
OPT_EBCDIC=	0, 1, 1, 0 X,	EBCDIC collating sequence
OPT_MULTI=	0, 2, 1, 0 X,	MULTINATIONAL collating sequence
OPT_NOSIGNAL=	0, 3, 1, 0 X,	Don't signal errors
OPT_SEQ_CHECK=	0, 4, 1, 0 X,	Sequence check on merge input
unused=	0, 5, 1, 0 X,	
OPT_NODUPS=	0, 6, 1, 0 X,	Delete records with duplicate keys
OPT_FIXED=	0, 7, 1, 0 X,	Records are fixed length (NYUsed)
OPT_LOCATE=	0, 8, 1, 0 X,	Use locate mode with RETURN_REC
OPT_LOAD_FILL=	0, 9, 1, 0 X,	Use LOAD_FILL on output file

## LITERAL

Values to index the sort statistics

```
! These are also used to define the global literals SOR$GK_STAT_XXX.
```

```
$EQUATE (STAT_K_, GBL, 0, 1,
(IDENT,)      Address of ASCII string for version number
(REC_INP,)    Records Input
(REC_SOR,)    Records Sorted
(REC_OUT,)    Records Output
(LRL_INP,)    LRL for Input
(LRL_INT,)    LRL of internal length record
(LRL_OUT,)    LRL for Output
(NODES,)      Nodes in sort tree
(INI_RUNS,)   Initial dispersion runs
(MRG_ORDER,)  Maximum merge order
(MRG_PASSES,) Number of merge passes
(WSEXTENT,)   Working-set extent
(MEM_USE,)    Memory usage
(WRK_ALQ,)    Work file usage
(DIRIO,)      Direct I/Os
(BUFIO,)      Buffered I/Os
(PAGEFLT,)    Page faults
(CPU_TIME,)   CPU time
(ELA_TIME,)   Elapsed time
(MBC_INP,)    MBC for Input
(MBC_OUT,)    MBC for Output
(MBF_INP,)    MBF for Input
(MBF_OUT,)    MBF for Output
(MAX_STAT,)): Last stat value
```

```
! Define a single key description in the key description buffer
```

```
$UNIT_FIELD
KBF_FIELDS =
SET
KBF_TYPE=      [XWORD],      ! Data type of key
KBF_ORDER=     [XWORD],      ! True iff descending order
KBF_POSITION=  [XWORD],      ! Offset to key within record (1..LRL)
KBF_LENGTH=    [XWORD]       ! Length of key
TES:
```

```
LITERAL KBF_K_SIZE = $FIELD_SET_UNITS;      ! Size in bytes
```

```
MACRO KBF_BLOCK = %EXPAND $UNIT_BLOCK(KBF_K_SIZE) FIELD(KBF_FIELDS) %;
```

```
! Define the key description buffer
```

```
MACRO KEY_NUMBER = 0, 0, 16, 0 %      ! Number of keys
KEY_KBF(N) = 2 + KBF_K_SIZE * (N), 0, 0, 0 %;
STRUCTURE KEY_BLOCK[O,P,S,E;BS=MAX_KEYS] =
[2 + KBF_K_SIZE*BS] (KEY_BLOCK + 0) <P,S,E>;
```

```
! Define the structure of a COLL_BLOCK, which is passed to SOR$SPEC_FILE
```

```
MACRO
```

SORLIB.REQ;1

16-SEP-1984 16:58:02.17<sup>D 12</sup> Page 13

COLL\_W\_LENGTH = 0, 0, 16, 0 %  
COLL\_B\_PAD = 3, 0, 8, 0 %  
COLL\_A\_PTAB = 4, 0, 32, 0 %

! Length of this block

SOR

! D  
! MAC

## COMMON INFORMATION

Information that must be available between calls to sort/merge is stored in a dynamically allocated data structure. The address of this data structure is stored in a context parameter that is passed to the sort/merge routines. If the context parameter is missing, the global variable SOR\$\$CONTEXT is assumed to contain this pointer.

## COMPILETIME

U\_ = 0;

## MACRO

U\_ = %ASSIGN(U\_+1)  
%NAME('U\_',%NUMBER(U\_)) %;      ! Macro to generate unique names

## LITERAL

COM\_K\_TREE=      13,      ! Number of longwords for TREE\_INSERT  
COM\_K\_SCRATCH=    10,      ! Number of longwords for SCRATCH\_IO  
COM\_K\_CDD=        2;      ! Number of longwords for CDD stuff

## \$FIELD

CTX\_FIELDS =  
SET

## Routines

COM\_COMPARE=      [XADDR],      ! Address of user comparison routine  
COM\_EQUAL=        [XADDR],      ! Address of equal-key routine  
COM\_INPUT=        [XADDR],      ! Address of input conversion routine  
COM\_OUTPUT=       [XADDR],      ! Address of output routine  
COM\_LENADR=       [XADDR],      ! Address of length, address routine  
COM\_NEWRUN=       [XADDR],      ! Address of new run routine  
COM\_ROUTINES=     [XDESC],      ! A dynamic string descriptor

## Storage for TREE\_INSERT

WM\_TREE\_INSERT=[%SUB\_BLOCK(COM\_K\_TREE)], ! Storage for TREE\_INSERT

## Global sort information

COM\_CTXADR=       [XLONG],      ! Address of users context Longword  
COM\_SORT\_TYPE=    [XBYTE],      ! Type of sort (TYP\_K\_RECORD,...)  
COM\_NUM\_FILES=    [XBYTE],      ! Number of input files  
COM\_WRK\_FILES=    [XBYTE],      ! Number of work files to use  
COM\_STABLE=       [%BIT],      ! Stable sort requested  
COM\_SEQ\_CHECK=    [%BIT],      ! Sequence check  
COM\_SIGNAL=       [%BIT],      ! Sort/merge should signal errors  
COM\_NOCHKPNT=     [%BIT],      ! Checkpointing should not be done  
COM\_LOAD\_FILL=    [%BIT],      ! Use load-fill on indexed files  
COM\_NODUPS=       [%BIT],      ! Delete records with duplicate keys  
U\_ =               [%BIT],      ! Use locate mode with RETURN\_REC

## Control flow flags

COM\_FLO\_SORT=      [%BIT],      ! May call Sort-Merge  
COM\_FLO\_NOINIT=   [%BIT],      ! May not call Pass-Files, Init-Sort or Init-Merge  
COM\_FLO\_RELEASE=   [%BIT],      ! May call Release-Rec  
COM\_FLO\_RETURN=   [%BIT],      ! May call Return-Rec or End-Sort



```

COM_FLO_DOMERGE=[$BIT],      ! May call Do-Merge
COM_FLO_ABORT=  [$BIT],      ! May only call End-Sort
:
: Flags to amend for V3 compatability hacks
COM_HACK_2ARGS= [$BIT],      ! Pass only 2 args to callback routines
COM_HACK_STRIP= [$BIT],      ! Strip the keys
:
: Merge-specific fields
: Note that COM_MRG_ORDER is non-zero iff this is a merge
COM_MERGE=      [$BIT],      ! Indicates a merge (not a sort)
COM_MRG_ORDER=  [XBYTE],      ! Order of the merge
:
: Spec text processing stuff
COM_SPEC_TKS=   [XWORD],      ! Size of keys portion of internal node
:
: Merge-specific fields
COM_MRG_INPUT=  [XADDR],      ! User-written merge input routine
COM_MRG_STREAM= [XLONG],      ! Stream number for stable merges
:
: Collating sequence stuff
COM_COLLATE=    [XADDR],      ! Addr of collating sequence routine
COM_ST_SIZ=     [XLONG],      ! Size (write-only)
:
: Key information
U =             [XADDR],      ! Address of key descriptions
COM_SPEC_FILE=  [XADDR],      ! Addr of structures from spec file
COM_TKS=        [XBYTE],      ! Total key size (as specified by user)
:
: Override flags - ignore the specification text for these options
COM_OVR_PROC=   [$BIT],      ! Process specified
COM_OVR_KEY=    [$BIT],      ! Key(s) specified
COM_OVR_CHKSEQ= [$BIT],      ! Check sequence specified
COM_OVR_STABLE= [$BIT],      ! Stable specified
COM_OVR_COLSEQ= [$BIT],      ! Collating sequence specified
COM_BS_DECM=    [$BIT],      ! Base sequence was DEC_MULTINATIONAL
U_ =            [BITS(4)],
:
: Counts
COM_RUNS=       [XWORD],      ! Current number of runs
COM_INP_RECNUM= [XLONG],      ! Input record number (stable & stats)
:
: Collating sequence information
COM_TIE_BREAK=  [$BIT],      ! Indicates tie-breaking
:
: Record format information

```

!no way  
!no way

```

COM_VAR=      [SBIT],      ! Flag indicating variable length input
U =           [SBITS(6)],
COM_MINVFC=    [XBYTE],      ! Length of VFC area in internal node
COM_MAXVFC=    [XBYTE],      ! Length of COM_RHB buffer
COM_FORMATS=   [XBYTE],      ! Number of different record formats
COM_LRL=       [XWORD],      ! Longest input record length
COM_SRL=       [XWORD],      ! Shortest record length
COM_LRL_INT=   [XWORD],      ! Length of internal format record
COM_LRL_OUT=   [XWORD],      ! Longest output record length
COM_RHB_INP=   [XADDR],      ! Address of VFC area (input side)
COM_RHB_OUT=   [XADDR],      ! Address of VFC area (output side)

```

#### File information

```

COM_PASS_FILES= [XADDR],      ! Output file characteristics
COM_OUT_DDB=    [XADDR],      ! Address of output file DDB
COM_INP_DDB=    [XADDR],      ! Address of input file DDBs
COM_INP_CURR=   [XADDR],      ! Address of current input file DDB
COM_INP_ARRAY=  [XADDR],      ! Array of input DDB pointers
COM_FILE_ALLOC= [XLONG],      ! File allocation specified by user
COM_SPC_DDB=    [XADDR],      ! Address of spec file DDB

```

#### Statistics information (used only for statistics)

```

COM_STAT_NODES= [XLONG],      ! Number of nodes in sort tree
COM_STAT_RUNS=  [XWORD],      ! Number of runs from dispersion
COM_STAT_PASSES= [XWORD],      ! Number of merge passes
COM_STAT_MERGE= [XBYTE],      ! Order of the merge
U =             [SBITS(24)],
COM_STAT_WS=    [XLONG],      ! Maximum WS used
COM_STAT_VM=    [XLONG],      ! Maximum VM used
COM_OMI_RECNUM= [XLONG],      ! Number of omitted records (for stats)
COM_OUT_RECNUM= [XLONG],      ! Output record number (for stats)

```

#### Storage for TREE\_INSERT

```

COM_TREE_LEN=  [XLONG],      ! Length of storage for tree
COM_TREE_ADR=  [XLONG],      ! Address of storage for tree

```

#### Scratch I/O information

```

COM_SCRATCH_IO= [$SUB_BLOCK(COM_K_SCRATCH)], ! Storage for SCRATCH_IO

```

#### Locking information

```

COM_LOCKED=    [XADDR],      ! List of locked code sections

```

#### Specification file stuff

```

COM_SPC_TXT=    [XDESC],      ! Dynamic string for spec file text

```

#### Specification file stuff

```

COM_RDT_SIZ=    [XBYTE],
COM_KFT_SIZ=    [XBYTE],
COM_CFT_SIZ=    [XBYTE],

```

```

COM_FDT_SIZE= [XBYTE],
COM_TDT_SIZE= [XBYTE],
COM_PAD= [XBYTE], ! Pad character
U = [ $BITS(16) ],
COM_RDT_ADR= [XADDR], ! Record definition table
COM_KFT_ADR= [XADDR], ! Key/data field table
COM_CFT_ADR= [XADDR], ! Constant field table
COM_FDT_ADR= [XADDR], ! Field definition table
COM_TDT_ADR= [XADDR], ! Test definition table
COM_CONST_AREA= [XADDR], ! Constant area (address)
COM_PTAB= [XADDR], ! Pointer to 256-byte table
U = [XADDR],
COM_WRK_SIZE= [XLONG], ! Length of work area
COM_WRK_ADR= [XADDR], ! Address of work area
COM_WRK_END= [XADDR], ! Address past end of work area

! Other stuff
COM_WORST= [XLONG], ! Worst error we've ever seen
COM_WF_NAMES= [ $COUNTED_LIST_OF_INDICES_INTO_CFT_OF_WORK_FILE_NAMES ],
[ $BYTES(1+MAX_WORK_FILES) ],
$ALIGN(FULLWORD)
COM_CDD= [ $SUB_BLOCK(COM_K_CDD) ], ! Storage for CDD stuff

! Additional storage for checkpoint stuff
COM_COUNTDOWN= [XLONG],
! Architectural flags (indicates which instructions are implemented)
COM_ARCHFLAG= [XLONG]
TES:

LITERAL
MACRO
CTX_K_SIZE= $FIELD_SET_SIZE; ! Size in longwords
CTX_BLOCK= BLOCK[CTX_K_SIZE] FIELD(CTX_FIELDS) %,
CTX_BLOCK_(S)= BLOCK[CTX_K_SIZE] FIELD(CTX_FIELDS,S) %;

%MESSAGE('CTX_K_SIZE = ', %NUMBER(CTX_K_SIZE))
UNDECLARE %QUOTE U_, U__:

```

## RECORD FORMATS

This section describes the various record formats that are used throughout Sort/Merge.

## INPUT RECORD FORMAT:

VAR (a word) is present only for variable length records  
 VFC is present only for VFC files  
 DATA is always present

## INTERNAL RECORD FORMAT:

FORM KEY VAR VFC DATA STAB       ! Record sort  
 FORM KEY RFA FILE STAB       ! Tag, address, index

VAR (a word) is present only for variable length records  
 VFC is present only for VFC files  
 KEY is present for keys or converted keys  
 FORM (a byte) is present only for multiple record formats  
 FILE (a byte) is present only for multi-file non-record sorts  
 STAB (a longword) is present only for stable sorts  
 RFA (RABSS\_RFA bytes) is present for non-record sorts

## OUTPUT RECORD FORMAT:

VAR VFC DATA       ! Record, tag sort  
 RFA FILE       ! Address sort  
 RFA FILE OKEY STAB       ! Index sort

VAR (a word) is present only for variable length records  
 VFC is present only for VFC files  
 FILE (a byte) is present only for multi-file non-record sorts  
 OKEY is the unconverted keys  
 STAB (a longword) is present only for stable index sorts

Assertions can be made on the following literals to determine the relative ordering of fields within a record.

## LITERAL

COM_ORD_RFA	= 0.	! RFA field
COM_ORD_FILE	= 1.	! File number field
COM_ORD_FORM	= 2.	! Format field
COM_ORD_OKEY	= 3.	! Original keys (for index sorts)
COM_ORD_STAB	= 4.	! Stable longword field
COM_ORD_KEY	= 5.	! Key or converted key field
COM_ORD_VAR	= 6.	! Length field
COM_ORD_VFC	= 7.	! VFC field
COM_ORD_DATA	= 8.	! Data field
COM_ORD_MAX	= 9.	! Largest order value



## DEVICE DESCRIPTION BLOCK

The DDB contains information for reading/writing a file. It does not contain all RMS structures, since the FAB, NAM, and other blocks may be discarded, thus decreasing the amount of virtual memory required.

```

$UNIT_FIELD
DDB_FIELDS =
SET
DDB_NEXT=      [XADDR],      ! Pointer to next DDB
DDB_NAME=      [$SUB_BLOCK(2)], ! File name length/address
DDB_IFI=       [XLONG],      ! Internal file identifier
DDB_FOP=       [XLONG],      ! File options
DDB_RAB_RAB=   [$BYTES(RABSC_BLN)], ! Record Access Block
DDB_FILE=      [XBYTE]      ! Input File number (0 on up)
TES;

LITERAL DDB_RAB=      %FIELDEXPAND(DDB_RAB_RAB,0);
UNDECLARE
LITERAL DDB_RAB_RAB;
LITERAL DDB_K_SIZE=   $FIELD_SET_UNITS;      ! Size in bytes
MACRO DDB_BLOCK=      %EXPAND $UNIT_BLOCK(DDB_K_SIZE) FIELD(DDB_FIELDS) %;

%MESSAGE('DDB_K_SIZE = ', %NUMBER(DDB_K_SIZE))

UNDECLARE
%QUOTE $DESCRIPTOR;

```

## LINKAGES

Several internal routines use JSB linkages to improve performance. Common linkages are defined here. Linkages to external routines are defined as LNK\_routine\_name.

LITERAL

```
COM_REG_SRC1 = 9
COM_REG_SRC2 = 10
COM_REG_CTX = 11;
```

MACRO

```
%PRESERVE(X)      = %NAME(X,'_PR') %
%NOPRESERVE(X)     = %NAME(X,'_NP') %
%NOTUSED(X)        = %NAME(X,'_NU') %
XREGMASK [P]       = 1^P %
REGMASK [ ]        = 0 OR XREGMASK_(%REMAINING) %;
```

KEYWORDMACRO

```
JSB_DEFN_(NAM,PM,GL,PR,NP,NU) =
```

LITERAL

```
%PRESERVE(NAM)      = REGMASK_(%REMOVE(PR)) + 0,
%NOPRESERVE(NAM)     = REGMASK_(%REMOVE(NP)) + 0,
%NOTUSED(NAM)        = REGMASK_(%REMOVE(NU)) + 0;
LINKAGE NAM = JSB(%REMOVE(PM)):
%IF NOT %NULL(GL) %THEN GLOBAL(%REMOVE(GL)) %FI
%IF NOT %NULL(PR) %THEN PRESERVE(%REMOVE(PR)) %FI
%IF NOT %NULL(NP) %THEN NOPRESERVE(%REMOVE(NP)) %FI
%IF NOT %NULL(NU) %THEN NOTUSED(%REMOVE(NU)) %FI
%;
```

JSB\_DEFN (

```
NAM = JSB_INPUT,           ! For COM_INPUT
PM = <REGISTER=COM_REG_SRC1,REGISTER=COM_REG_SRC2>,
PR = <COM_REG_SRC2>,
NP = <0,1,2,3,4,5,6,COM_REG_SRC1>, ! R6 holds the variable length
NU = <7,8>,
GL = <CTX=COM_REG_CTX> );
```

JSB\_DEFN (

```
NAM = JSB_NEWRUN,          ! For COM_NEWRUN
NU = <4,5,6,7,8,10>,
NP = <0,1>,
PR = <2,3,6>,
GL = <CTX=COM_REG_CTX> );
```

JSB\_DEFN (

```
NAM = JSB_COMPARE,        ! For COM_COMPARE
PM = <REGISTER=COM_REG_SRC1,REGISTER=COM_REG_SRC2>,
PR = <COM_REG_SRC1,COM_REG_SRC2>,
NP = <0,1,2,3,4,5>,
NU = <6,7,8>,              ! Really???
GL = <CTX=COM_REG_CTX> );
```

JSB\_DEFN (

```
NAM = JSB_OUTPUT,         ! For COM_OUTPUT
PM = <REGISTER=COM_REG_SRC2>,
```

```
PR = <COM_REG_SRC2>,
NU = <7,8,9>
NP = <0,1,2,3,4,5,6>
GL = <CTX=COM_REG_CTX> );
```

! R6 needed???

```
JSB_DEFN (
  NAM = JSB_EQUAL,
  PM = <REGISTER=COM_REG_SRC1,REGISTER=COM_REG_SRC2>,
  PR = <COM_REG_SRC1,COM_REG_SRC2>,
  NP = <0,1>,
  NU = <2,3,4,5,6,7,8>,
  GL = <CTX=COM_REG_CTX> );
```

```
JSB_DEFN (
  NAM = JSB_LENADR,
  PM = <REGISTER=COM_REG_SRC2;REGISTER=0,REGISTER=1>,
  PR = <COM_REG_SRC2>,
  NP = <0,1>,
  NU = <2,3,4,5,6,7,8,9>,
  GL = <CTX=COM_REG_CTX> );
```

```
JSB_DEFN (
  NAM = JSB_INSERT,
  PM = <STANDARD>,
  PR = <7,8>,
  NP = <0,1,2,3,4,5,6,COM_REG_SRC1,COM_REG_SRC2>,
  GL = <CTX=COM_REG_CTX> );
```

```
JSB_DEFN (
  NAM = JSB_READINS,
  PM = <REGISTER=6,REGISTER=8>,
  PR = <7,8>,
  NP = <0,1,2,3,4,5,6,9,10>,
  GL = <CTX=COM_REG_CTX> );
```

```
JSB_DEFN (
  NAM = JSB_EXTRACT,
  PM = <STANDARD>,
  PR = <7,8>,
  NP = <0,1,2,3,4,5,6,COM_REG_SRC1,COM_REG_SRC2>,
  GL = <CTX=COM_REG_CTX> );
```

```
LINKAGE
  CAL_ACCESS = CALL ( STANDARD;
                      REGISTER=0;
                      REGISTER=1);
                      GLOBAL(CTX=COM_REG_CTX);
```

```
LINKAGE
  CAL_CTXREG = CALL: GLOBAL(CTX=COM_REG_CTX);
```

## TUNING PARAMETERS

These values are used to tune the sort.

```

LITERAL
TUN_K_NONTREE = 192,      ! Number of pages to not use for the tree
TUN_K_FALLBACK = 64,      ! Minimum pages for tree for a large sort
TUN_K_CALC_FI = TRUE,     ! True to calculate FI in sort tree
TUN_K_CALC_FE = TRUE,     ! True to calculate FE in sort tree
TUN_K_OUT_PREALL = TRUE,   ! True to preallocate output file
TUN_K_WRK_PREALL = FALSE,  ! True to preallocate work files
TUN_K_ALIGN_NODE = 2,      ! Log2 of alignment for nodes (longword align)
TUN_K_ALIGN_TREE = 9,      ! Log2 of alignment for sort tree (page align)
TUN_K_MRG_COST = 0,        ! Cost of merge
TUN_K_PURGWS = FALSE,     ! True to purge working set before INIT_TREE
TUN_K_LCK_CTX = TRUE,      ! True to lock context area in WS
TUN_K_LCK_TREE = 3,        ! Pages of tree to lock in WS
TUN_K_LCK_CODE = TRUE,     ! True to lock code in WS
TUN_K_BINMOVE = 32,        ! Max number of bytes to move with binary moves
TUN_K_MAX_MERGE = 20;      ! Maximum merge order for internal merges

MACRO
TUN_K_BUFSIZE =
  %IF NOT HOSTILE_ELAN
  %THEN 50 * COM_K_BPERPAGE      ! Bytes in a buffer
  %ELSE 5 * COM_K_BPERPAGE      ! Bytes in a buffer
  %FI %;

LITERAL
FUN_K_CHECKPOINT = FALSE;      ! True to generate code for checkpointing
ASSERT(TON_K_MAX_MERGE GEQ MAX_MERGE_ORDER)

%IF NOT FUN_K_CHECKPOINT
%THEN
  UNDECLARE %QUOTE COM_NOCHKPNT, %QUOTE COM_COUNTDOWN;
%FI

```



## SRT

STR

MAC

UNDECLARED  
SORS\_FACILITY;

UNDECLARE  
SORS FACILITY:

881

**NACRÒ**

LITERAL

```

NAME
DEFshr (
    BADLOGIC, SEVERE,
    CLOSEDEL, ERROR,
    CLOSEIN, ERROR,
    CLOSEOUT, ERROR,
    INSVIRMEM, SEVERE,
    OPENIN, SEVERE,
    OPENOUT, SEVERE,
    READERR, ERROR,
    SYSERR, SEVERE,
    TEXT, WARNING,
    WRITEERR, ERROR);

```

```
! The following macro is used to diagnose an unrecoverable error, instead of
! calling SORSEERROR directly.
```

## MACRO

```

SOR$$FATAL(X) = (RETURN SOR$$ERROR(
(X) AND NOT STSM SEVERITY OR STSK SEVERE
%IF %LENGTH GTR 1- %THEN , %REMAINING %FI)) %:

```

T  
T

## T E X T U A L   I N F O R M A T I O N

User-visible text is defined here. This text may be translated or changed, subject to the restrictions described below.

Default file extension

```
MACRO
STR_DEF_EXT =      '.DAT' %;
```

Default specification file, and default specification file extension

```
MACRO
STR_DEF_SPECFILE =  'SYS$INPUT' %,
STR_SPC_EXT =      '.SRT' %;
```

These macros define the external and internal representations of options for command line qualifiers. The first parameter in each pair may be translated; the second, however, is used to define internal name for this option, and may not be translated.

```
MACRO
STR_OPT_OUTFMT =      ! outfile/FORMAT=(...)
'FIXED',              'FIXE',
'VARIABLE',           'VARI',
'CONTROLLED',         'CONT',
'SIZE',               'SIZE',
'BLOCK_SIZE',         'BLOC' %,

STR_OPT_INPFMT =      ! inpfiler/FORMAT=(...)
'FILE_SIZE',         'FILE',
'RECORD_SIZE',       'RECO' %,

STR_OPT_PROCESS =     ! /PROCESS=...
'RECORD',            'RECO',
'TAG',               'TAG',
'ADDRESS',           'ADDR',
'INDEX',             'INDE' %,

STR_OPT_KEY =         ! /KEY=...
'ASCENDING',         'ASCE',
'BINARY',            'BINA',
'CHARACTER',         'CHAR',
'DECIMAL',           'DECI',
'DESCENDING',        'DESC',
'UNSIGNED',          'UNSI',
'F_FLOATING',        'F_FL',
'D_FLOATING',        'D_FL',
'G_FLOATING',        'G_FL',
'H_FLOATING',        'H_FL',
'LEADING_SIGN',      'LEAD',
'NUMBER',            'NUMB',
'OVERPUNCHED_SIGN', 'OVER',
'POSITION',          'POSI',

! NUMBER:nn
! POSITION:nn
```

```
MACRO STR_LOG_WORKFILE = 'SORTWORK' %.
```

SORLIB.REQ;1

16-SEP-1984 16:58:02.17 Page 26

STR\_LOG\_WORKNUM = '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ' %;

! Default file name string to use for the work files.

MACRO

STR\_DEF\_WORKFILE = 'SYS\$SCRATCH:SORTWORK.TMP' %;

SR1

!L1

! A  
! XIF  
! XTH  
! XEL  
! XF1

MAC



## CLEAN-UP ROUTINES

Clean-up routines are called by SOR\$SEND\_SORT. To facilitate information-hiding, the following mechanism is used. It allows each sub-system to declare a clean-up routine to clean up its data structures (so that SOR\$SEND\_SORT need not know the format of the data structures, or even the name of the clean-up routine).

A clean-up routine is declared by:

```
FORWARD ROUTINE CLEAN_UP;  
SOR$SEND_ROUTINE (CLEAN_UP);  
ROUTINE CLEAN_UP: CAL_CTXREG NOVALUE = ...
```

```
MACRO SOR$SEND_PSECT (X) = %NAME(%EXACTSTRING(30,'_', 'SOR$RO_CODE'),X) %;  
MACRO SOR$SEND_ROUTINE (X) =  
PSECT NODEFAULT= %EXPAND SOR$SEND_PSECT_(2)(PIC,SHARE,NOWRITE,EXECUTE);  
OWN %NAME('_',X): PSECT(%EXPAND SOR$SEND_PSECT_(2))  
INITIAL(X-%NAME('_',X)) %;
```

## EXEC - MODE VARIANT

A variant of Sort/Merge is made available to the RDMS group for use in EXEC mode. This is gotten by compiling the following modules with the /VARIANT=1 command qualifier. Note that the /VARIANT qualifier will have no effect when compiling the require files. External references from these modules are named SOR\$fac\$name. For example, the following code would be in SORINTERF.

```
%IF HOSTILE
%THEN
    MACRO
        LIB$GET_VM = SOR$LIB$GET_VM %,
        LIB$FREE_VM = SOR$LIB$FREE_VM %;
%FI
```

Another variant of Sort/Merge is made available for JRD on ELAN. This variant is gotten by compiling with /VARIANT=3. The major distinction between this and the previous is that the address of the context longword passed to Sort/Merge is passed to several of the SOR\$fac\$name system services.

The following modules are needed for these variants:  
COM.REQ, SORLIB.REQ, OPCODES.REQ, SORMSG.MSG, SORINTERF.B32,  
SORKEYSUB.B32, SORSORT.B32, SORSCRIO.B32, SORFILNAM.B32

```
MACRO HOSTILE = %VARIANT %;
MACRO HOSTILE_ELAN = (%VARIANT AND %VARIANT^-1) %;
```

SORLIB.REQ;1

16-SEP-1984 16:58:02.17 Page 29

! End of SORLIB.REQ

SR1

XIF

L1

STF

MAC

XFI



DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY